MaRMI-III: A Methodology for Component-Based Development

Dong-Han Ham, Jin-Sam Kim, Jin-Hee Cho, and Su-Jung Ha

As component-based development (CBD) rapidly spread throughout the software industry, a comprehensive methodology is needed to apply it more systematically. For this purpose, a new CBD methodology named Magic & Robust Methodology Integrated III (MaRMI-III) has been developed. The purpose of this paper is to present MaRMI-III by its constituent processes and claim that it can be used to support system developers conduct CBD in a consistent manner. First, we review the CBD approach to system development and the role of CBD methodology, and then we explain the several characteristics of MaRMI-III which are considered necessary to the CBD environment. Next, we explain a process model of MaRMI-III which separates the development process from the project management process and prescribes well-ordered activities and tasks that the developer should conduct. Each phase forming the Process Model is explained in terms of its objectives and main constituent activities. Some techniques and workproducts related to each phase are also explained. Finally, to examine the usefulness of MaRMI-III, an analytical comparison with other CBD methodologies and the results of a questionnaire survey are described.

Keywords: Component-based development (CBD), software component, and systems development methodology.

Manuscript received Mar. 25, 2003; revised Nov. 4, 2003.

This work was supported by the Korea Ministry of Information and Communication.

Dong-Han Ham (phone: +82 42 860 1658, email: dhham@etri.re.kr), Jin-Sam Kim (email: jinsam@etri.re.kr), Jin-Hee Cho (email: chojh@etri.re.kr), and Su-Jung Ha (email: hsj@etri.re.kr) are with Basic Research Laboratory, ETRI, Daejon, Korea.

I. Introduction

Component-based development (CBD) has been considered the most viable approach for realizing software reuse and thus for efficiently developing high-quality software-based systems [1], [2]. CBD aims to develop software systems by assembling reusable software components and modifying them if necessary [3]. Such an approach is expected to bring about several advantages. One primary benefit is that it significantly enhances software reusability, reliability, and maintainability, as well as reducing time-to-market, thereby increasing the productivity of the software development process [3], [4]. Additionally, a component platform framework or architecture provides components with various services such as transaction, security, and persistency, so that a high level of quality and reliability is guaranteed to component-based systems [5]. Finally, several management activities, including quality assurance and maintenance, potentially become easier by taking a component as the unit of management and, consequently, diminishing managing complexity [6].

A software component can be defined in various ways according to its abstraction level or range of use [2], [4]. However, the most generally accepted definition is that of D'Souza et al.: "An independently deliverable unit of software that encapsulates its design and implementation and offers interfaces to the outside, by which it may be composed with other components to form a larger whole" [7]. Another frequently quoted definition is that of Szyperski, which points out the important characteristics of a component: "A unit of composition with contractually specified interfaces and explicit dependencies only. A component can be deployed independently and is subject to third-party composition" [8].

CBD is divided into two processes: component development (CD) and component-based software development (CBSD) [9]. CD focuses on how to build highly reusable independent software modules (design for reuse), whereas CBSD mainly strives to construct a system by composing the available components that best meet the users' requirements and technological constraints (design with reuse).

In general, when a new development paradigm emerges and there is a lack of experience, a development methodology is needed. A methodology is defined as a collection of processes specifying activities to develop a system and techniques to be used for the activities throughout the life cycle [10]. From the developers' cognitive perspective, the most significant value of a methodology is that it allows them to "do the right things" as well as to "do the things right."

ETRI conducted a questionnaire survey of scores of Korean software companies about the actual practices of development methodology in 2001 [11]. The survey results indicated that 97% have a strong need for methodology in constructing software-based systems. In particular, 66.3% answered that they were willing to introduce a CBD methodology, whereas 12.6% of the respondents were interested in a structured and information-engineering methodology and 19% were interested in an object-oriented methodology. These results indicate the evident need and importance of a CBD methodology. Moreover, the lessons learned from conducting CBD demonstrated that a CBD methodology providing a well-organized process and various techniques is indispensable for the success of a CBD project [12].

Currently, well-known CBD methodologies include the Rational Unified Process (RUP) [13], Catalysis [7], Select Perspective [14], UML Component [15], Compuware's Uniface [16], and Castek CBD [17]. Of these, RUP and Catalysis are the most widely used in real industry. RUP is a well-defined software engineering process and provides a customizable process framework. Though it is not a methodology specific to a CBD project, one of its essential principles is to build systems with components, and thus RUP can be used or customized to support CBD projects. The most remarkable feature of RUP is that it has two dimensions, a time dimension showing a life cycle and a discipline dimension indicating certain activities to be primarily conducted along the life cycle. The main advantage of using RUP is that the various tools provided from Rational Software support activities specified in RUP, enabling iterative and incremental development. Catalysis provides a very comprehensive process from business modeling to code and process patterns adopting various development requirements. However, it is not a thorough methodology, rather a semi-structured set of design principles, advice, and patterns throughout the life cycle. Such a characteristic makes it difficult for the developer to view the big picture when using it. Additionally, the lack of management activities is a crucial shortcoming of Catalysis.

However, in spite of their strong points, none of these methodologies give a complete solution to CBD [10], [18]. This is because the CBD approach is fairly new and has a greater complexity in both the development and project management processes compared to the prior development paradigms. One of their shortcomings is that they handle problems mainly in the implementation and deployment phases, instead of within the full-system life cycle [19]. Also, the use of some methodologies is dependent on specific software tools or organizations, resulting in a lack of generality [10]. A third shortcoming is that most of CBD methodologies are weak in supporting architecture modeling and software reuse [20]. Finally, from our experience, we notice that there is a lack of a detailed process enabling developers to work in a procedural way, regardless of their level of expertise in CBD. From the cognitive perspective of developers, procedural task is more effective and less cognitively burdensome in most situations of problem-solving such as design activity [21]. These shortcomings highlight the need for a new methodology that will achieve the claimed CBD benefits.

To effectually support developers in conducting a CBD project, a methodology should be comprehensive, user-friendly, and customizable. In other words, it should address all kinds of development and management activities, providing specific guidance on those activities as well as allowing developers to customize it to their projects. Another point to consider is that it should address design problems specific to a broadly used component technology platform such as Java 2 Platform, Enterprise Edition (J2EE) or .NET. In addition, new concepts such as component, interface, and architecture should be clearly defined and represented [20]. With this view in mind, we have developed a new CBD methodology named Magic & Robust Methodology Integrated III (MaRMI-III)[1].

The purpose of this paper is to present MaRMI-III and claim that it can be used to support system developers in conducting CBD. First, this paper describes the characteristics of MaRMI-III, and then it explains the detailed process in connection with its other constituent elements. Lastly, evaluation studies to validate the usefulness of MaRMI-III are described.

1) MaRMI is a series of development methodologies. MaRMI-I, the first methodology, was developed on the basis of a structured development and information engineering concept in 1997. Afterwards, MaRMI-II, which is an object-oriented methodology particularly aimed at the information and communication industry, was developed in 1998. The phonetic spelling of MaRMI means cutting out in Korean [22].

II. Outline of the Methodology

When developing MaRMI-III, we collaborated with six active software companies that have a lot of experience in CBD projects. Also, we made reference to the previous CBD methodologies introduced in section I and attempted to draw useful contents from them, with consideration of our experience. Thus, our methodology is the result of combining our CBD experience with the quality practices of previous CBD methodologies and with other CBD-related literature.

In this section, we discuss the several characteristics of MaRMI-III. Firstly, MaRMI-III makes use of Unified Modeling Language (UML) as a modeling notation. UML, which most system developers are accustomed to using, is regarded as a de facto standard notation for object-oriented modeling [23], [24]. For this practical reason, in MaRMI-III, system analysis and design activities are conducted by using UML diagrams. In particular, MaRMI-III takes a use-case driven approach. Use-case can be effective for identifying user requirements and specifying components. It is also a basic unit for incremental development.

Secondly, MaRMI-III is architecture-centric and thus lays emphasis on system architecture design in the early phase of the development process. A system architecture designed to be stable and extensible in the early phase guarantees high reusability of software components. In MaRMI-III, architecture is organized into three layers: technical architecture, software architecture, and component architecture. In general, technical architecture provides the technical environment that developers should understand and consider. As will be explained in detail later, MaRMI-III deals with designing a software architecture that is a collection of software components, and a component architecture that is a collection of business components concerned with specific business processes or functions.

Thirdly, MaRMI-III addresses both the development of systems from components and the development of a component itself. To realize such systems, developers should not only use off-the-shelf components but also build components for themselves if there are no relevant components in the off-the-shelf market [25]. Regarding this, what is noteworthy is a mini-project that uses a time box as a management method for realizing use cases. A mini-project is a small-scale work unit that develops one component or more in a sequential or concurrent way. It enables system developers to take a stepwise and incremental approach. This results in several benefits: minimizing project risk, managing requirements with the participation of users, giving developers continuous motivation, and so on.

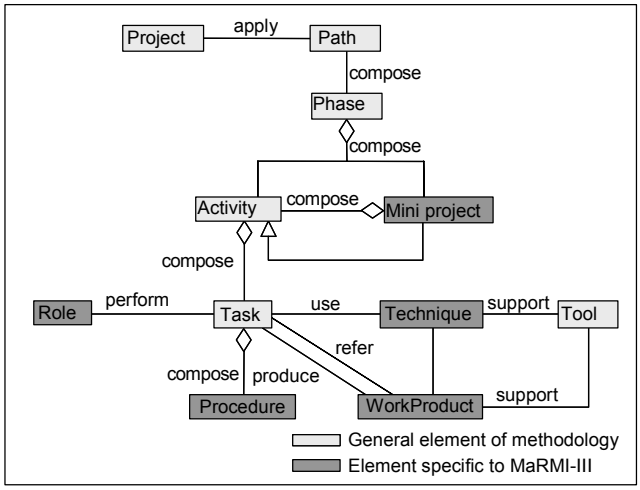

Fig. 1. Metamodel of MaRMI-III.

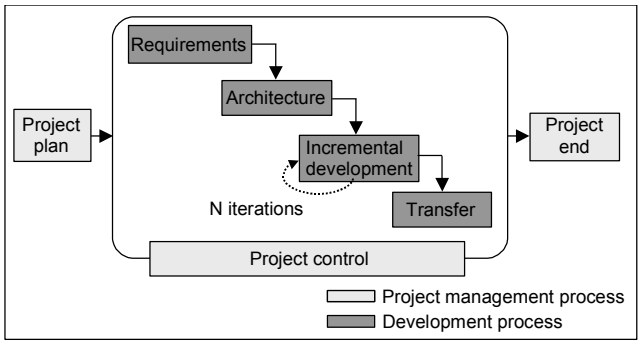

Fig. 2. Underlying concept of the overall process.

Fourthly, the metamodel of MaRMI-III, as shown in Fig. 1, is reasonably compatible with the Software Process Engineering Metamodel, specified by the Object Management Group [26]. It gives a description of what elements constitute methodology and how they are related to each other. The definition of each element is given in Appendix A.

Fifthly, Fig. 2 shows the underlying concept of the process of MaRMI-III. What is noted here is the division of the development and project management processes. This reflects the view that the management of a CBD process is so complex that it needs to be dealt with separately. The project management process is comprised of three phases: *Plan, Control*, and *End*. The development process consists of four phases: *Requirements, Architecture, Incremental Development*, and *Transfer*. Each of these four phases goes with the *Control* phase from its beginning to end. In the phase of *Incremental Development*, an iterative approach through mini-projects is taken to implement the system in a step-wise way. Figure 3 details the overall process to the level of activity. For example, the *Requirements* phase is made up of three activities: requirements understanding, requirements definition, and development strategy set-up.

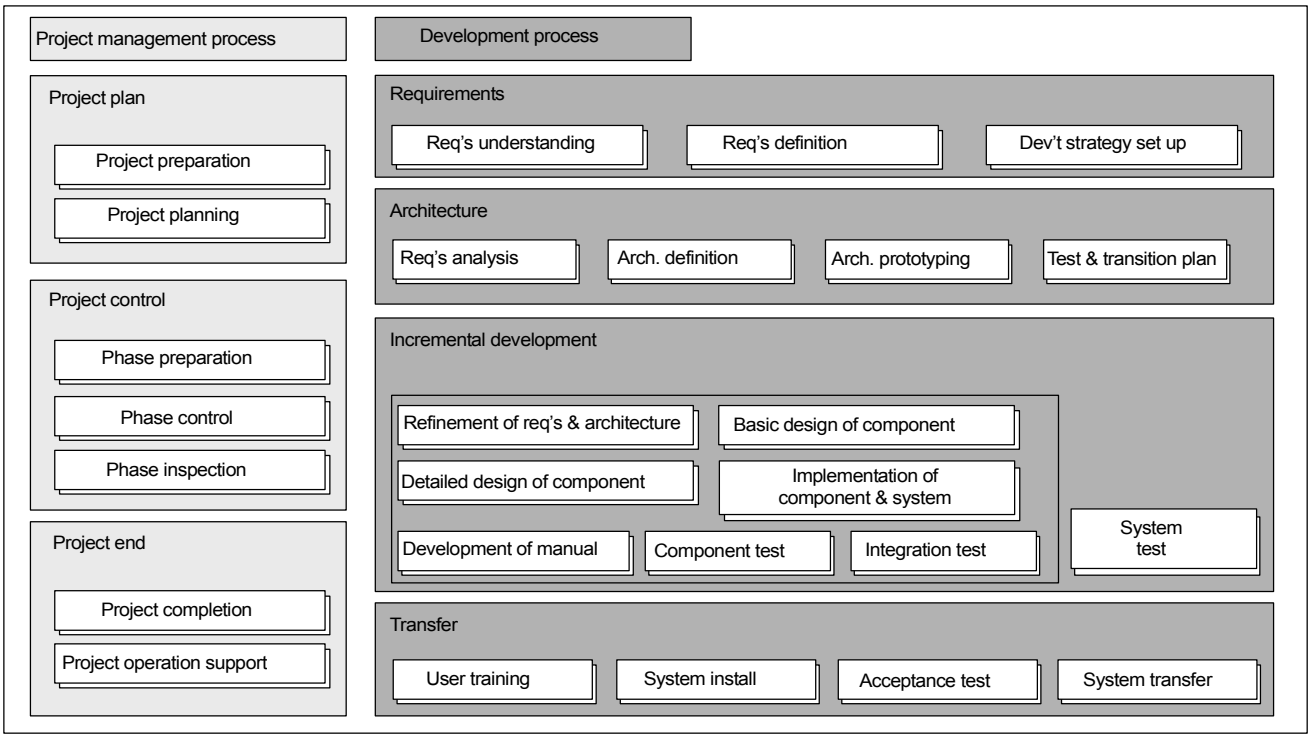

Fig. 3. Detailed process to the level of activity.

Sixthly, MaRMI-III is made up of three major elements: a Process Model, a Set of Techniques, and a Set of WorkProducts. The Process Model prescribes what activities and tasks constitute the development and project management processes and how they should be conducted and in what order. As an example, Fig. 4 shows the configuration diagram of the *Incremental Development* phase. The Set of Techniques helps developers conduct tasks specified in the Process Model in an efficient way. Examples of Techniques include object modeling, cost and benefit analysis, design pattern, architecture style, and the Architecture Tradeoff Analysis Method. A workproduct is a description of a piece of information or physical entity used or produced by the tasks. To aid developers in writing the prescribed workproducts, MaRMI-III provides a Set of WorkProducts that predefine its format and some contents. The content of these workproducts are interrelated. Figure 5 illustrates such an example in the phase of *Incremental Development*. Additionally, MaRMI-III provides two example case studies: the bidding management system in e-commerce and the external cooperation system in the banking industry, which will be used as references for its application.

Finally, MaRMI-III has two versions, each of which addresses the different component technologies, platform-J2EE and .NET. Currently, both of them are the most extensively used platforms in the software component industry. The main difference between the two versions exists mainly in the task of detailed component design and implementation of the component and system.

In the next section, each phase forming the overall process is explained in terms of objective, constituent activities, and associated main-techniques and workproducts.

III. Development Process

The *Requirements* phase is the first step of the development process. This phase aims at collecting and identifying users' requirements in consideration of the system vision. If necessary, a business model is created so that the system background can be understood more accurately. The boundary of the system to be developed is clarified with a use-case diagram and conceptual model. Designers verify the adequacy of the requirements by developing a user interface (UI) prototype and testing it with the participation of users. An initial draft of the system architecture is defined on the basis of the requirements, use-case diagram, and conceptual model. Reusable components addressing some parts of the requirements are examined to enhance the productivity of the development process. As is widely acknowledged, identification of the correct requirements becomes increasingly important to the success of system development, not to mention CBD. For this, MaRMI-III emphasizes the utilization of the use-case diagram, which represents the system's functionality and can be served as a basic unit for requirements identification, management, and testing. Additionally, several techniques for user-centered

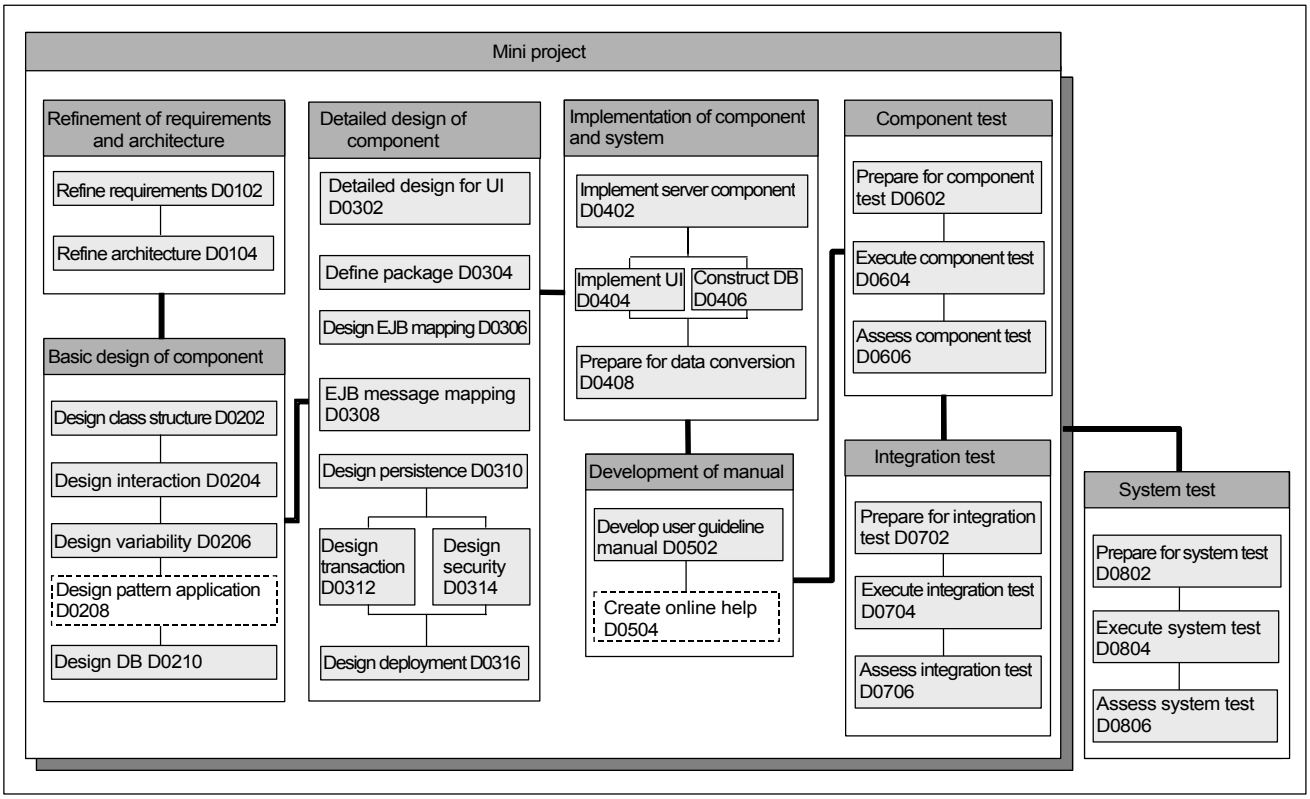

Fig. 4. Configuration diagram of incremental development phase (J2EE version).

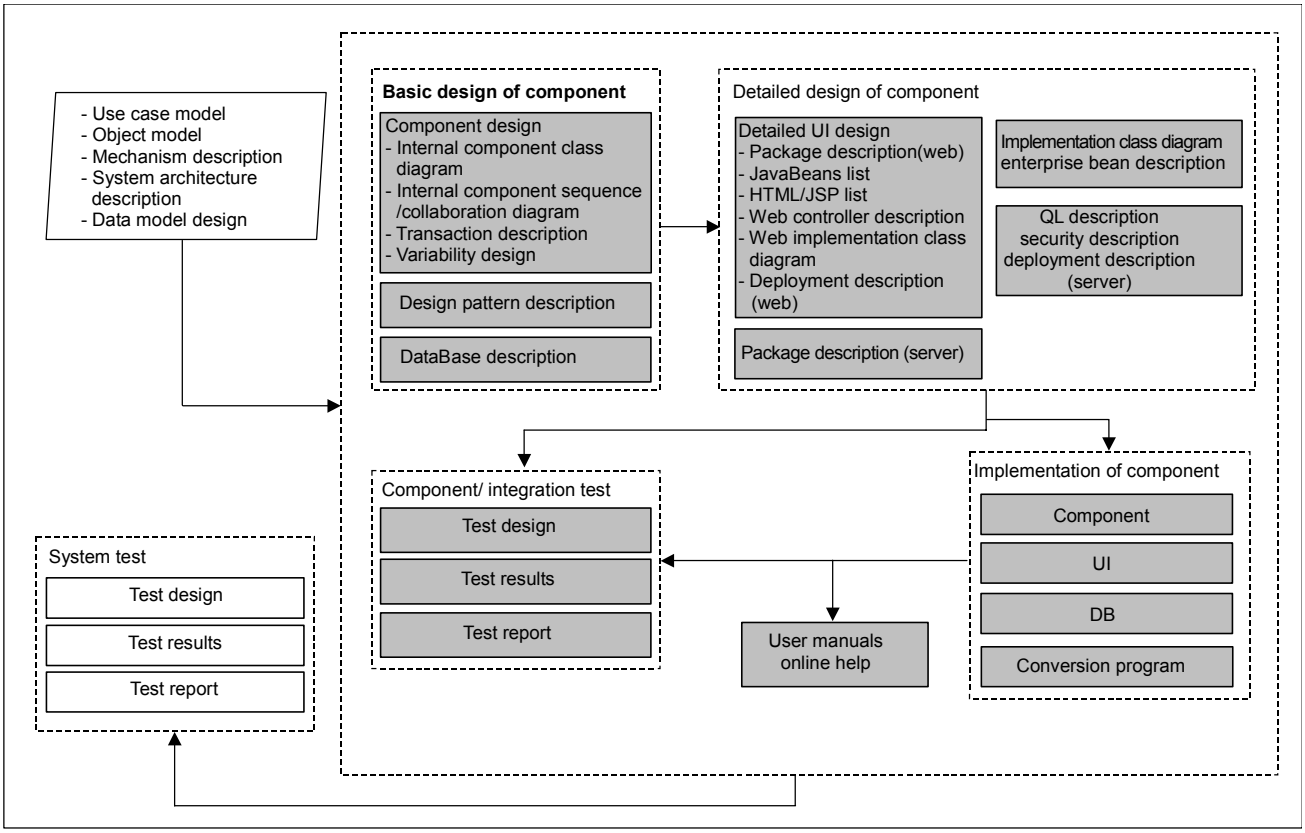

Fig. 5. Relation between workproducts of incremental development phase (J2EE version).

design, such as brainstorming, structured interview, and questionnaires are used for capturing the right user's requirements. In this phase, the main workproducts include a vision description, requirements collection, business use-case model, use-case model, and UI prototype.

Next is the *Architecture* phase, of which the objective is to define a stable system architecture accommodating efficient software reuse. As previously explained, system architecture is defined by three architectures: technical architecture, software architecture, and component architecture, as shown in Fig. 6. Technical architecture means the given technical environment for component execution, which includes hardware and a network. Developers seldom consider this architecture as a design item, but they must understand it as such to effectively use it. Therefore, the primary concern of developers is how to design software architecture and component architecture. Software architecture is a set of software components. Its design focuses on the reusability of software components and stability against a change of users' needs. A use-case model can be effectively used to understand the design requirements of software architecture. Component architecture is a set of business components that deals with certain kinds of business tasks in the work domain. It is designed so that business components can reflect users' needs flexibly and quickly. Comparing software architecture with component architecture again, the former is related to an implementation perspective, whereas the latter relates to the business process and tasks. To design the system architecture, the user's requirements are first refined by structuring a use-case diagram and creating a system-object model. Using the refined use-case model, designers define the software architecture, satisfying all kinds of requirements as well as the identified quality attributes. A detailed design mechanism and strategy to implement the defined software architecture are derived, and a logical data model is designed at this time. On the basis of the logical data model, developers identify the interface in consideration of the cohesion of business tasks and refine the reference relations between interfaces in consideration of the software architecture and technical architecture. Next, they derive the final business components and detailed component operations during the process of refining the interface and prepare the component specifications by component unit. The derived components are then structuralized to create a component architecture model. After the software architecture and component architecture are created, the components are packaged and structuralized, and the initial system architecture made out in the *Requirements* phase is then refined and finally defined.

Examination and obtainment of reusable components is done after the final system architecture is built. The design pattern

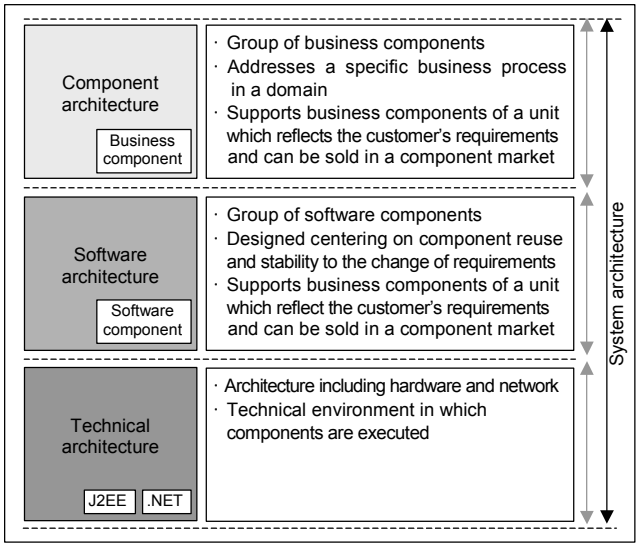

Fig. 6. System architecture of three layers.

and several diagrams of UML, such as a collaboration diagram, class diagram, and sequence diagram, are mainly used as techniques for architectural design. As constructing a system architecture may involve unpredictable risk factors, designers should strive to get rid of such probable risk factors through the architecture prototype. The workproducts that should be necessarily produced from this phase include a software architecture specification, mechanism description, component specification, system architecture description, and an architecture prototype.

As described above, MaRMI-III suggests that software architecture be designed first, and component architecture be designed later. However, in the early phases of component-based system development, business components of component architecture are implicitly considered for designing software architecture. As a design progresses, developers can identify refined business components independent of software architecture. Here, designing software architecture requires a top-down approach because developers should decompose system functions, referring to mainly use-case models, as well as a bottom-up approach because the software components ought to be designed in consideration of implementation issues and the technical architecture. The design of the component architecture also needs a hybrid approach since developers should keep in mind the overall purpose and process of the needed systems as well as software components implementing business components.

In the phase of *Incremental Development*, the component-based system is implemented through a mini-project based on the use-cases and a system architecture created in previous phases, as shown in Fig. 5. In general, though it is not mandatory, it is suggested that one mini-project should aim at

developing one component using two to five designers. The mini-project allows developers to implement a system in a repeatable and incremental way, thereby minimizing the risk of project failure. Through conducting mini-projects, use-cases and the system architecture are refined from the implementation viewpoint. Classes and components are designed using various diagrams of UML, such as class, sequence, and activity diagrams. Then, components are designed in detail, together with a database and a UI, taking the component technology platform into consideration. After integrating the designed components, an integration and system test is executed to verify whether the system runs as designed. The important workproducts produced from this phase are the UI-detailed design description, component-detailed design description, database design description, refined system-architecture description, component code, and user's manual. In the case of the J2EE version, the component-detailed design description includes a package description (server), implementation class-model description, enterprise JavaBeans description, transaction design description, Query Language definition description, security definition description, and a deployment description (server-J2EE). Contrastingly, the .NET version involves a namespace definition description, application definition description, service-component definition description, Structured Query Language definition description, DLL definition description, and deployment description (server-.NET).

Finally, the *Transfer* phase is conducted to store the developed components into a component repository or to install the developed system into a user's environment. If a repository or system is already being used, it is converted into a new one for smooth operation. The final acceptance should be gained, checking if the developed component or system is compatible with the users requirements. Then, all kinds of developed outputs are delivered to the users. The system installation report, acceptance test report, user's training report, and system observation report are the main workproducts in this phase.

IV. Project Management Process

In general, the project management process is made up of various activities to accomplish the purpose of the project, meeting the goals of delivery time, cost, and quality. Its activities and their focus are dependent on the technological environment and the specific characteristics of the system development process. Additionally, as a system development process becomes more complex, its management should accordingly become more systematic and comprehensive. With this view in mind, MaRMI-III provides a project management process separated from the development process, which reflects several typical features of a CBD-based project and gives special emphasis on quality assurance and time management in the mini-project.

As the first step of the project management process, the *Plan* phase has the objective of obtaining the acknowledgement of conducting a project and preparing the many things needed for system development. When necessary, a contract is made on the service agreement. This phase addresses the activities of both a project sponsor and a project performer and their interactions to make a project contract. Additionally in this phase, a detailed plan for developing the system is also made out on the basis of the project contract. The workproducts created after conducting this phase include the project draw-up, document requesting proposal, project proposal, project contract-document, project-working plan, and quality control plan.

The objective of the *Control* phase is to support the commencing of each phase in the development process, manage its progress, and conduct quality control on the workproducts. Specifically, this phase involves the following activities: preparing for launching upon each phase in the development process; establishing detailed development activities and tasks; preparing a standard for managing a project's progress; establishing a plan for conducting mini-projects; monitoring and evaluating the project's progress, conducting quality control activities to ensure the quality of workproducts; and adjusting the project schedule and plan to optimize the performance of the project by checking the results of each phase in the development process. The main workproducts are a phase working plan, mini-project working plan, project progress report, mini-project inspecting report, phase inspecting report, and quality control report.

After completing all the development processes, the *End* phase evaluates the final results of the project and makes a report on them. Workproducts produced from the project are arranged for reuse in the future. In this phase, the project manager settles accounts for the finished project to do cost accounting. Finally, project resources, such as personnel and equipment, are relocated and a maintenance plan is established to give a satisfactory quality-in-use to users. A project completion report, system operation plan, and system operation contract are the main workproducts of this phase.

V. Evaluation of the Methodology

The effectiveness of a new system can be validated in several ways [27], [28]. In general, evaluation methods can be divided into three categories: formal experiment, case study, and survey [28]. In the case of the methodology, formal experiment and

full case study seem to be difficult because they require too much time and cost and show a lack of sound metrics. For this reason, we compared MaRMI-III with other CBD methodologies according to some evaluation criteria and subsequently supplemented the comparison results by conducting a questionnaire survey of organizations having experience in using MaRMI-III. In the following, we will describe the subjective comparison results.

As shown in Table 1, MaRMI-III has attractive advantages in most of the evaluation criteria. As in the case of the other methodologies, it uses UML as a notation and stresses an iterative and incremental approach as well as an early prototyping. Its primary advantage is that it provides a highly specific process, task, and procedure covering the full development and project management life cycle, which developers can customize to be suitable for their work context. For example, Fig. 7 illustrates what activities compose the *Control* phase and what tasks constitute those activities. The number at the bottom-right corner of the box indicates the identification number of each activity and task. In more detail, Fig. 8 shows the detailed procedure of the task of internally inspecting the mini-project. From this figure, we can find that there are three roles involved in this task: the project manager, quality manager, and component developer. Other information includes the workproducts that are used for input to this task or produced from this task. To further understand which input workproducts are used for which procedure, which procedure produces which workproducts, which roles are involved in which procedures, and what specific technique can be used for this task, developers can consult the contents of the Process Model which is a main element of MaRMI-III. They can also use a set of techniques and workproducts that enable them to do tasks in a more procedural and systematic way, thereby improving the performance of the development and management.

In addition to the above, there are other benefits from the use of MaRMI-III. For instance, it deals well with architectural design problems which are gradually becoming more critical to the success of CBD projects. Architectural design and evaluation problems in CBD are too complex to rely on the developer's intuition and experience. The number of factors comprising the problems and their interrelations form a broad design space and thus place a high cognitive load on the developer. The detailed and proved procedures and techniques of MaRMI-III would be a merit in that they can reduce the architectural design space appropriately.

As software is increasingly becoming interactive, a user interface design needs a more systematic process and more guidance. Compared to the other methodologies, MaRMI-III gives much more attention to the design process of the user interface over the development life cycle. However, it does not provide many of the user interface design and evaluation

Table 1. Comparison of MaRMI-III with other methodologies.

Criteria	Catalysis	RUP	Select perspective	MaRMI-III
Availability	Book web site training	Book web site training	Book web site training	Book web site training
Tool support	●	●	●	×
Component development process	×	●	●	●
Component-based software development process	●	●	●	●
Project management process	×	○	○	●
Quality management process	×	○	×	○
Guidance on development and management	○	○	○	●
Workproduct template and guidance	×	○	○	●
Guidance on identifying component	●	●	●	●
Method for component specification	●	●	●	●
Guidance on component technology platform	×	×	×	●

(●: Fully supported, ○: Partially supported, ×: Not supported)

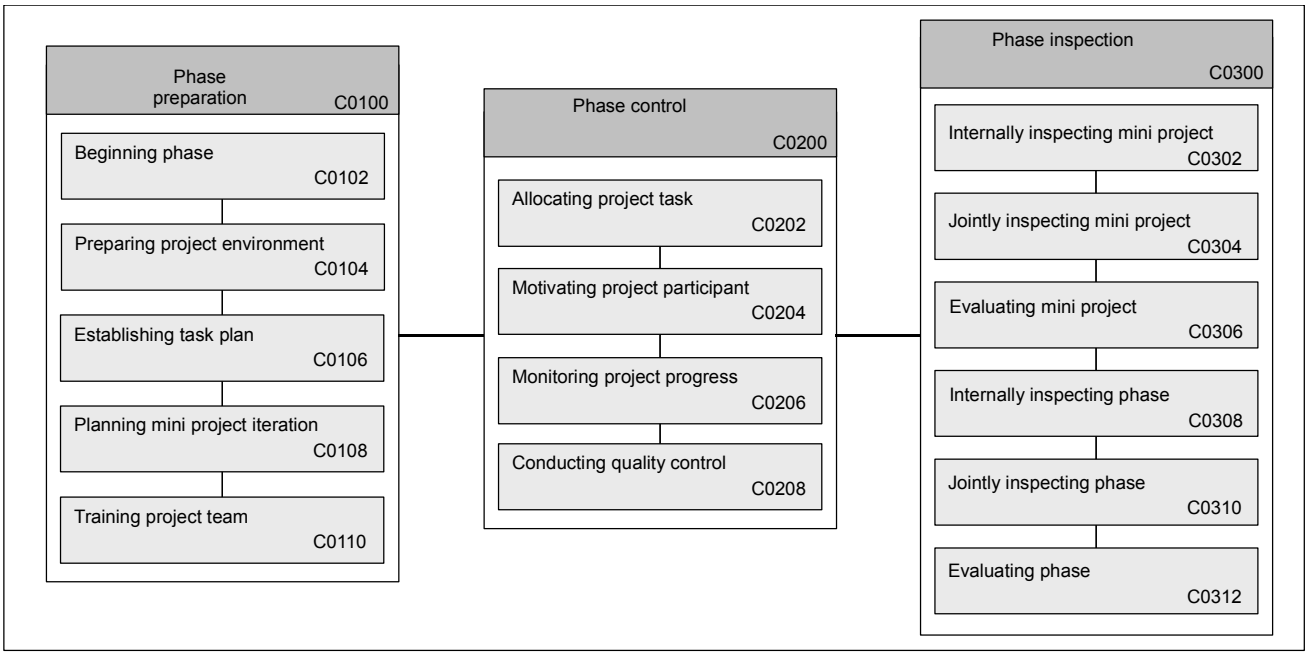

Fig. 7. Configuration diagram of control phase.

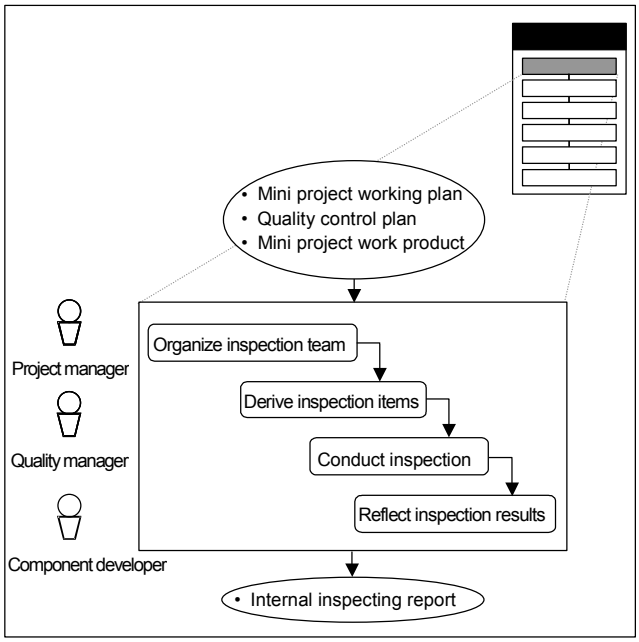

Fig. 8. Procedural diagram of internally inspecting mini project task.

methods that are widely used in the industry.

To investigate the effectiveness and quality of MaRMI-III, its J2EE version was reviewed and validated by five international experts. All of them are professors in universities in the USA and have a lot of experience in research and consulting on CBD. The review method they used is a combined type of expert review with heuristic evaluation. Thus, they separately reviewed MaRMI-III with a short checking criteria and came together to discuss their review results. The criteria used are as follows: how well the state-of-the-art J2EE technologies are reflected, to what degree MaRMI-III is complete and consistent in its contents and structure, and how practicable and usable it is for developers to apply. The overall results indicate that MaRMI-III would be a good map of guidance for developing software components and component-based software, though some shortcomings and problems were identified. The typical problems pointed out include inconsistency of terms, obscurity in the definition of a few terms, information redundancy, lack of detailed explanation for a few activities and tasks, and so on. All of them were reflected in revising the J2EE version and in making the .NET version.

There are several software process standards such as ISO/IEC 12207 [29], ISO/IEC 15504 [30], CMMI [31], and so forth. They are classified into two groups by their main purpose. One group is concerned with a common framework for a software life-cycle process (e.g., ISO/IEC 12207), and the other group provides a framework to assess and improve the software process (e.g., ISO/IEC 15504 and CMMI). To examine the coverage of the processes provided by MaRMI-III, we compared them to those of ISO/IEC 12207 and ISO/IEC 15504. To summarize the comparison, MaRMI-III addresses most of the processes prescribed in those standards except the following: the operation process (5.4, CUS 4)[2], maintenance process (5.5, ENG 2), audit process (6.7, SUP 7), and improvement process (7.3, ORG 2). However, software

2) The first item in the parentheses indicates the process of ISO/IEC 12207 and the second is for ISO/IEC 15504

process standards only prescribe the requisite processes and their workproducts. They do not define the detailed procedure, guidance and technique used to execute the processes. In this regard, MaRMI-III can be used as a concrete and practical methodology to support adopting and conforming to the software process standards.

From the analytical evaluation, we can also identify the weak points of MaRMI-III. First, specific guidance on testing related activities is insufficient. For example, the problem of how to use component testing results in integration, and system testing needs to be supplemented. Second, MaRMI-III does not suggest an adequate number of iterations in the phase of *Incremental Development* according to the characteristics of the project context. Third, although a metamodel is provided, a more comprehensive metal-level framework specifying the roles of the methodology elements and concepts, as well as how their relations are specified, needs to be developed. Such kind of framework would be a good map for guiding the customization of the methodology.

Although subjective assessment has its own merit, it is necessary to collect empirical data on the actual use for a better guarantee of the benefits of MaRMI-III. In the current situation, a survey is regarded as the most desirable empirical method, taking into consideration our available resources. Thus, we conducted a questionnaire survey of organizations having experienced using MaRMI-III. So far, twenty-seven organizations have used MaRMI-III and they were asked to respond to our questionnaire survey by email. The questionnaire includes 63 questions, some of which are sub-categorized or similar to others. The type of question is either open or closed, according to its characteristics. In the closed question, requiring only one choice, responses were measured on a 5-point semantic-difference scale, which ranges from "very good" to "very poor". Eleven responses were obtained, resulting in a response rate of about 41%. This rate can be considered above average in survey research, though our sample size is not large [32]. However, one response was considered insufficient for the analysis, so ten responses were analyzed. Appendix B illustrates the profile of the organizations and the main results.

The survey results are summarized as follows. The profiles of the respondents vary in their organization type and size, and the purpose of using MaRMI-III is dependent on their working purposes, as shown in Table B-1. The period of their experience in CBD and use of MaRMI-III also varies, as shown in Table B-2. First, the perceived overall quality of MaRMI-III was questioned on two points: on content and organization, and on its completeness. Most of the respondents selected the "above-average" level on both points, as illustrated in Table B-3. As to the advantages and shortcomings, several features were evenly selected, and can be viewed in Tables B-4 and B-5. Detailed procedures and workproduct templates are comparatively pointed out as the main advantages of MaRMI-III. However, the detailed procedures result in a large-volume working manual, and this can be troublesome to some people. This is supported from the data which shows that such a large volume was selected most as the main shortcoming. Table B-6 shows that the process model was generally considered the best element of MaRMI-III, and of all of the elements of MaRMI-III was also chosen as the worst element one or more times. The completeness and detailedness of the process model ranked in the level, "above average" shown in Table B-7. But, regarding the component identification and specification, we didn't get a favorable answer, as can be seen in Table B-8. These seem to be the weak points of MaRMI-III, and should be revised and improved in the future. Next, we asked about the usefulness of three points featuring MaRMI-III, which are included in Tables B-9 thru B-11. Seventy percent of the respondents graded the usefulness of contents on specific platform technologies in the level "good". All of them gave a favorable score on the usefulness of the separation of development and project management. However, the project management part nearly scored in the "average" level. Finally, all of the respondents rated as "above average" the effectiveness of MaRMI-III on their work. To sum up the survey results, MaRMI-III has a potential to aid CBD developers in conducting their projects.

VI. Conclusion

MaRMI-III has been developed to support software developers taking a CBD approach by providing a coherent streamlined Process Model and a Set of Techniques and WorkProducts. Specifically, it provides a well-defined development process that is compatible to Software Process Engineering Metamodel at a meta-level and a project management process that emphasizes quality and risk aspects. Its users are also aided by specific techniques informing procedural ways to deal with certain tasks or problems specified in the Process Model, as well as workproduct collections providing a workproduct template and guidelines for writing them. Both English and Korean versions of MaRMI-III are available. MaRMI-III deals with the design problems related to J2EE and .NET, both of which are widely used for a CBD-technology platform. Evaluation studies for the effectiveness of MaRMI-III showed that it could usefully support the work of CBD developers.

However, based on the evaluation results, several things remain as a matter to be further studied in order to make MaRMI-III more usable. First, it is unreasonable to apply any

kind of processes and activities to any kind of development project. Thus, it is advisable for developers to discreetly select processes and activities appropriate to their own work. The problem of how to selectively apply MaRMI-III according to the inherent characteristics of the project such as size, constraints in terms of cost and schedule, and a priority of quality criteria should be further studied. Second, when using a heavy-weight methodology like MaRMI-III, users find it difficult to understand the interrelation between workproducts and to trace the information flowing through them. To lessen a user's cognitive load in such tasks, some types of guidelines or maps should be devised. Finally, to obtain more practical results on the strengths and weaknesses of MaRMI-III, continuous evaluations should be made. Although the evaluation studies we conducted have their own advantages, they have limits in revealing the practical matters in using MaRMI-III. Thus, it will be necessary to use more powerful and long-term validation techniques, such as project monitoring, field studies, and a synthetic approach for future evaluations.

References

[1] A.W. Brown and K.C. Wallnau, "The Current State of CBSE," *IEEE Software*, Sept./Oct. 1998, pp. 37-46.

[2] W. Hasselbring, "Component-Based Software Engineering," in S.K. Chang (ed.): *Handbook of Software Engineering and Knowledge Engineering*, vol. 2, World Scientific Publishing, New Jersey, 2002, pp. 289-305.

[3] I. Crnkovic and M. Larsson, *Building Reliable Component-Based Software Systems*, Artech House, 2002.

[4] P. Brereton and D. Budgen, "Component-Based Systems: a Classification of Issues," *IEEE Software*, Nov. 2000, pp. 54-62.

[5] L. Brownsword, T. Oberndorf, and C.A. Sledge, "Developing New Processes for COTS-Based Systems," *IEEE Software*, July/Aug. 2000, pp. 48-55.

[6] L.A. Maciaszek, *Requirements Analysis and System Design: Developing Information Systems with UML*, Addison-Wesley, 2001.

[7] D.F. D'Souza and A.C. Willis, *Object, Components, and Frameworks with UML: The Catalysis Approach*, Addison-Wesley, 1998.

[8] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley, 1998.

[9] W-J. Lee, O-C Kwon, M-J. Kim, and G-S. Shin, "A Method and Tool Support for Identifying Domain Components Using Object Usage Information," *ETRI J.*, vol. 25, no. 2, 2003, pp. 121-132.

[10] Z. Stojanovic, A.N.W. Dahanayake, and H.G. Sol, "A Methodology Framework for Component-Based System Development Support," *Proc. of the 6th CaiSE/IFIP8.1 Int'l Workshop on Evaluation of Modeling Methods in Systems Analysis and Design EMMSAD'01*, 2001.

[11] *Project Research Plan Report: Development of Component-based Development Methodology*, ETRI, 2001.

[12] M. Sparling, "Lessons Learned through Six Years of Component-Based Development," *Comm. of the ACM*, vol. 43, no. 10, 2000, pp. 47-53.

[13] P.B. Kruchten, *The Rational Unified Process: An Introduction(2nd Ed.)*, Addison-Wesley, 2000.

[14] Select Perspective, URL: http://www.selectbs.com

[15] J. Cheesman and J. Daniels, *UML Components: A Simple Process for Specifying Component-Based Software,*" Addison-Wesley, 2001.

[16] Compuware's Uniface Methodology, URL: http://www.compuware.com/products/uniface

[17] Castek, URL: http://www.castek.com

[18] N. Boertien, M.W.A. Steen, and H. Jonkers, "Evaluation of Component-Based Development Methods," *Proc. of the Sixth Int'l Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*, 2001.

[19] J.Q. Ning, "Component-Based Software Engineering (CBSE)," *Proc. of Fifth Int'l Symp. on Assessment of Software Tools and Technologies*, June 1997.

[20] J.Q. Ning, "A Component-Based Software Development Model," *Proc. of 20th Int'l Computer Software and Applications Conf.*, Aug. 1996.

[21] Wickens C.D., *Eng. Psychology and Human Performance (2nd Ed.)*, Harper Collins Publishers, New York.

[22] D-H. Ham, J. S. Kim, J. H. Cho, and S. J. Ha, "Developing a Methodology for Component-Based Development," *Proc. of 2002 Asia-Pacific Industrial Eng. and Management Science Conf.*, 2002

[23] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language: User Guide*, Addison-Wesley, 1999.

[24] C. Kobryn, "Modeling Components and Frameworks with UML," *Comm. of the ACM*, vol. 43, no. 10, 2000, pp. 31-38.

[25] I. Crnkovic, "Component-Based Software Engineering-New Challenge in Software Development," *Software Focus*, vol. 2, no. 4, 2002.

[26] *Software Eng. Process Metamodel Specification (final adopted specification)*, OMG, Dec. 2001.

[27] M.V. Zelkowitz and D.R. Wallace, "Experimental Models for Validating Technology," *IEEE Computer*, May 1998, pp. 23-31.

[28] B.A. Kitchenham, S.G. Linkman, and D.T. Law, "Critical Review of Quantitative Assessment," *Software Eng. J.*, March 1994, pp. 43-53.

[29] ISO 12207: *Information Technology-Software Life Cycle Processes*, Int'l Organization for Standardization, 1995.

[30] ISO 15504: *Information Technology-Software Process Assessment (Part 1 to 9)*, Int'l Organization for Standardization. 1998.

[31] CMMI-SE/SW: *CMMI for Systems Eng. and Software Eng. V 1.1, Continuous Representation*, Software Eng. Institute, 2001.

[32] B.A. Kitchenham and S.L. Pfleeger, "Principles of Survey Research-Part 2: Designing a Survey," *Software Eng. Notes*, vol. 27, no. 1, 2002, pp. 18-20.

Appendix A: Elements of Metamodel

Elements constituting metamodel are defined as follows.

Phase The highest level of work which is a structured set of *Activities*.

Activity A structured set of *Tasks* which are logically interconnected. Every activity has checkpoints with which the project manager judges how well a project progresses.

Task The smallest unit of work which developers should accomplish and consists of more than one *Procedure*.

Procedure The lowest level of work specifying the order by which one *Task* should be conducted.

Mini Project A kind of *Activity* which produces a reusable component and system under a limited length of time.

Technique A method which can be used to accomplish a *Task* and is based on a special procedure, concept, and skill.

Tool A means which can be used to efficiently accomplish a *Task* and is usually a type of computer-based software.

Role Organization or people who conduct a *Task* for a project.

WorkProduct All kinds of outputs which are produced as a result after accomplishing a *Task*.

Appendix B: Survey Results

Table B-1. The profile of respondents (multiple choices are possible).

Type \ Size	0 ~ 50	50 ~ 100	100 ~ 300	Over 300
S/W development	2	1	0	0
SI and IT consulting	4	1	0	2
University	1	1	0	0
Etc	0	0	0	0

Table B-2. The period of their experience on CBD and MaRMI-III (for CBD, only applied to the first and second types in Table B-1).

	CBD		MaRMI-III	
	n	%	*n*	%
Below 1 month	0	0	2	20
1 ~ 3 months	0	0	2	20
3 ~ 6 months	1	10	1	10
6 ~ 12 months	3	30	1	10
12 ~ 24 months	1	10	4	40
Over 2 years	4	40	0	0

Table B-3. The perceived overall-quality of MaRMI-III.

	Contents and organization		Completeness of CBD methodology	
	n	%	*n*	%
Very good	1	10	1	10
Good	6	60	5	50
Average	3	30	1	10
Poor	0	0	3	30
Very poor	0	0	0	0

Table B-4. The perceived advantages of MaRMI-III (multiple choices are possible).

	n	%
Detailed procedures and its logicalness	6	24
A lot of techniques	2	8
Detailed workproduct templates	7	28
Example case studies	3	12
Manual written in Korean	3	12
Separation of development and project management processes	4	16
Making overall CBD process more easily understandable	0	0

Table B-5. The perceived shortcomings of MaRMI-III (multiple choices are possible).

	n	%
Large volume due to detailed procedures	4	30
Small number of techniques	1	8
Difficulty of using workproduct templates	1	8
Insufficient example case studies	3	23
Incorrect expression of contents	1	8
Difficulty due to separation of development and project management process	0	0
Lack of tool support	3	23

Table B-6. The best and worst element of MaRMI-III.

	Best element		Worst element	
	n	%	*n*	%
Process model	6	60	3	30
Set of techniques	2	20	1	10
Set of workproducts	2	20	3	30
Example case studies	0	0	2	20
Nothing	0	0	1	10

Table B-7. The completeness and detailedness of a process model of MaRMI-III.

	Completeness		Detailedness	
	n	%	*n*	%
Very good	0	0	1	10
Good	6	60	3	30
Average	3	30	5	50
Poor	1	10	1	10
Very poor	0	0	0	0

Table B-8. Contents on component identification and specification.

	Identification		Specification	
	n	%	*n*	%
Very good	0	0	0	0
Good	2	20	3	30
Average	4	40	6	60
Poor	4	40	1	10
Very poor	0	0	0	0

Table B-9. Usefulness of contents on specific platform technologies.

	Usefulness	
	n	%
Very good	0	0
Good	7	70
Average	2	20
Poor	1	10
Very poor	0	0

Table B-10. Usefulness of separation of development and project management processes.

	Usefulness of separation	
	n	%
Very good	1	10
Good	9	90
Average	0	0
Poor	0	0
Very poor	0	0

Table B-11. Usefulness of project management.

	Usefulness of project management	
	n	%
Very good	0	0
Good	3	30
Average	7	70
Poor	0	0
Very poor	0	0

Table B-12. Overall effectiveness of MaRMI-III on their work.

	Usefulness of project management	
	n	%
Very good	2	20
Good	3	30
Average	5	50
Poor	0	0
Very poor	0	0

Dong-Han Ham received the BS in industrial engineering in 1993 from INHA University, and the MS and PhD in industrial engineering in 1995 and 2001 from KAIST (Korea Advanced Institute of Science and Technology). He is a Senior Researcher in ETRI (Electronics and Telecommunications Research Institute), Daejeon, South Korea. His research areas include software engineering and information systems, human-computer interaction, and cognitive systems engineering. He has performed several works related to information display design in complex systems, software user interface, software quality testing and certification, and component-based systems development. He is now performing a research project concerning an embedded systems development framework and standardization of a software architecture and product line.

Jin-Sam Kim is a Principle Member of the Engineering Staff at ETRI, Korea. He received the MS in computer science from Chung-Ang University, Korea. His research interests include software process modeling and improvement in software development. He is now performing a research project concerning an embedded systems development framework and the standardization of a software architecture and product line.

Jin-Hee Cho received the BS and MS in computer engineering in 1992 and 1996 from Kyungpook National University. He is a Senior Researcher in ETRI, Daejeon, South Korea. His research areas include software engineering and information systems. He has performed several works related to object-based systems development and component-based systems development. He is now performing a research project concerning an embedded systems development framework and product line.

Su-Jung Ha received the BS in computer engineering in 1991 from Myeong-Ji University and the MS in computer science in 2001 from Korea University. She is a Senior Researcher in ETRI, Daejeon, South Korea. Her research areas include software engineering, software development methodology and software quality. She has performed several works related to software development methodology, software quality evaluation and component-based systems development methodology. She is now performing a research project concerning an embedded systems development framework.